Towards Weakly Consistent Local Storage Systems

Ji-Yong Shin\textsuperscript{1,2}, Mahesh Balakrishnan\textsuperscript{2}, Tudor Marian\textsuperscript{3}, Jakub Szefer\textsuperscript{2}, Hakim Weatherspoon\textsuperscript{1}

\textsuperscript{1}Cornell University, \textsuperscript{2}Yale University, \textsuperscript{3}Google
Consistency/Performance Trade-off in Distributed Systems

- Slower reads to latest version of data
- Faster reads to stale data using weak consistency

Diagram:
- Primary
- Back-up
- Replication
- Clients

ACM Symposium on Cloud Computing
## Server Comparison

<table>
<thead>
<tr>
<th>Year</th>
<th>2006</th>
<th>2016</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model (4U)</td>
<td>Dell PowerEdge 6850</td>
<td>Dell PowerEdge R930</td>
</tr>
<tr>
<td>Memory</td>
<td>64GB</td>
<td>6TB</td>
</tr>
<tr>
<td>Network</td>
<td>2 × 1GigE</td>
<td>2 × 1GigE 2 × 10GigE</td>
</tr>
<tr>
<td>Storage</td>
<td>8 × SCSI/SAS HDD</td>
<td>24 × SAS HDD 10 x PCIe SSD</td>
</tr>
</tbody>
</table>

Modern Server ≈ Distributed System
Can we apply distributed system principles to local storage systems to improve performance?

Consistency and performance trade-off
Why Consistency/Performance Trade-off?

<table>
<thead>
<tr>
<th>Distributed Systems</th>
<th>Modern Servers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Different versions of data exist in different servers due to network delays for replication</td>
<td>Different versions of data exist in different storage media due to logging, caching, copy-on-write, deduplication, etc.</td>
</tr>
<tr>
<td>Older versions are faster to access when the client is closer to the server</td>
<td>Older versions are faster to access when they are on faster storage media</td>
</tr>
</tbody>
</table>

Reasons for different access speeds
- RAM, SSD, HDD, hybrid-drives, etc.
- Disk arm contention
- SSD under garbage collection
- Degraded mode in RAID
Fine-grained Log and Coarse-grained Cache

- Multiple logged objects fit in one cache block
Goal

• Speedup local storage systems using stale data (consistency/performance trade-off)

  – How should storage systems access older versions?
  – Which version should be returned?
  – What should be the interface?
  – What are the target applications?
Rest of the Talk

• StaleStore

• Yogurt: An Instance of StaleStore

• Evaluation

• Conclusion
StaleStore

• A local storage system that can trade-off consistency and performance
  – Can be in any form
    • KV-store, filesystem, block store, DB, etc.
  – Maintains multiple versions of data
    • Should have interface to access older versions
  – Can estimate cost for accessing each version
    • Aware of data locations and storage device conditions
  – Aware of consistency semantics
    • Ordered writes and notion of timestamps and snapshots
    • Distributed weak (client-centric) consistency semantics
StaleStore: Consistency Model

- Distributed (client-centric) consistency semantics
  - *Per-client, per-object* guarantees for reads
  - Bounded staleness
  - Read-my-writes
  - **Monotonic-reads**: A client reads an object that is the same or later version than the version that was last read by the same client
StaleStore: Target Applications

- Distributed applications
  - Aware of distributed consistency
  - Can deal with data staleness

- Server applications
  - Can provide per client guarantees
Rest of the Talk

• StaleStore

• Yogurt: An Instance of StaleStore

• Evaluation

• Conclusion
Yogurt: A Block-Level StaleStore

- An log-structured disk array with cache [Shin et al., FAST’13]
  - Prefer to read from non-logging disks
  - Prefer to read from the most idle disk
Yogurt: Basic APIs

- **Write (Address, Data, Version #)**
  - Versioned (time-stamped) Write
  - Version # constitutes snapshots

- **Read (Address, Version #)**
  - Versioned (time-stamped) Read

- **GetCost(Address, Version #)**
  - Cost estimation for each version
Yogurt Cost Estimation

- GetCost(Address, Version) returns an integer

- Disk vs Memory Cache
  - Cache always has lower cost
    (e.g. cache = -1, disk = positive int)

- Disk vs disk
  - Number of queued I/Os with weights
  - Queued writes have higher weight than reads
Reading blocks from Yogurt

- **Monotonic-reads example**

  Client session
  Lowest Ver = 3
  Read version [Blk 1: Ver 5]

- Read block 1
  1. Checks current timestamp: highest Ver = 8
  2. Issues GetCost() for block 1 between versions 3 and 8 (N queries with uniform distance)
  3. Reads the cheapest: e.g. 1 (5): Read(1, 5)
  4. Records version for block 1

![Diagram showing client session and cache access]

Global Timestamp

8

Cache

...
Data construct on Yogurt

- High level data constructs span multiple blocks
  - Blocks should be read from a consistent snapshot
  - Later reads depend on prior reads: GetVersionRange()
Rest of the Talk

• StaleStore

• Yogurt: An Instance of StaleStore

• Evaluation

• Conclusion
Evaluation

- Yogurt: 3 disk setting with memory cache
- Focus on read latency while using monotonic-reads
- Clients simultaneously access servers
- Primary-backup setting

Baseline 1: reads latest data in the primary server

\[100\text{ms delay}\]

Baseline 2: reads latest data in a local server

Utilize stale data in a local server
Evaluation: Block Access

- Uniform random workload
- 8 clients access one block at a time
- X-axis: # of available older versions built up during warm up
Evaluation: Key-Value Store

- YCSB Workload-A (Zipf with 50% read, 50% write)
- 16 clients access multiple blocks of key-value pairs
- KV Store “greedily” searches the cheapest using Yogurt APIs
- KV pairs can be partially updated
Conclusion

• Modern servers are similar to distributed systems

• Local storage systems can trade-off consistency and performance
  – We call them **StaleStores**
  – Many systems have potentials to use this feature

• Yogurt, a block level StaleStore
  – Effectively trades-off consistency and performance
  – Supports high level constructs that span multiple blocks
Thank you

Questions?
Extra slides
Fine-grained log and coarse-grained cache

• Multiple logged objects fit in one cache block
Fine-grained log and coarse-grained cache

- 8 threads reading and writing at 9:1 ratio
- KV-pairs per cache block from 2 to 16
- Allowed staleness from 0 to 15 updates (bounded staleness)

![Graph showing average read latency vs. allowed staleness for different numbers of items in cache blocks]
Deduplicated system with read cache

• Systems that cache deduplicated chunks
Deduplicated system with read cache

- 8 threads reading and writing at 9:1 ratio
- Deduplication ratio controlled from 30 to 90%
- Allowed staleness from 0 to 15 updates (bounded staleness)

![Graph showing read latency against allowed staleness for different deduplication ratios (30%, 50%, 70%, 90%). The highest max latency is for 30%.]
Write cache that is slow for reads

- Griffin: disk cache over SSD for SSD lifetime

<table>
<thead>
<tr>
<th>Disk Cache</th>
<th>SSD</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 (5)</td>
<td>0 (3)</td>
</tr>
<tr>
<td>1 (2)</td>
<td>1 (1)</td>
</tr>
<tr>
<td>1 (3)</td>
<td>2 (0)</td>
</tr>
<tr>
<td>3 (4)</td>
<td>3 (4)</td>
</tr>
<tr>
<td>...</td>
<td>Addr (Ver)</td>
</tr>
</tbody>
</table>

Logged blocks

- Slow read (latest)
- Fast read (stale)
- Flushes latest

Linear block address space
Write cache that is slow for reads

- 8 threads reading and writing at 9:1 ratio
- Data flushed from disk to SSD every 128MB to 1GB writes
- Allowed staleness from 0 to 7 updates (bounded staleness)